테이블 기반 알고리즘을 이용한 CRC8의 구현

서석배* 김영선* 박종억* 공종필* 용상순* 이승훈**

CRC8 Implementation using Direct Table Algorithm

Seok-Bae SEO* Young-Sun KIM* Jong-Euk PARK* Jong-Phil KONG*
Sang-Soon YONG* and Seung-Hoon LEE**

Abstract

CRC (Cyclic Redundancy Codes) is a error detection method for the date transmission, which is applied to the GRDDP (GOES-R Reliable Data Delivery Protocol) between satellite and GEMS (Geostationary Environmental Monitoring Sensor) on the GEO-KOMPSAT 2B development. This paper introduces a principle of the table based CRC, and explains software implementation results of the CRC8 applied to GEMS.

초 록

CRC (Cyclic Redundancy Codes)는 데이터 전송 시 오류 발생 유무를 검출하기 위한 하나의 방법으로, 정지궤도복합위성(GEO-KOMPSAT 2B) 개발에서는 정지궤도환경탑재체와 위성 간의 GRDDP (GOES-R Reliable Data Delivery Protocol)에 사용되고 있다. 본 논문에서는 CRC를 구현할 때 널리 사용하는 테이블 기반 CRC의 원리를 소개하고, 이를 기반으로 환경탑재체 개발에서 사용 중인 CRC8을 소프트웨어로 구현한 결과를 설명한다.

키워드 : CRC (Cyclic Redundancy Codes), 테이블 기반 CRC (CRC direct table algorithm), CRC 소프트웨어(CRC software), SpaceWire 통신(SpaceWire communication), and GRDDP (GOES-R Reliable Data Delivery Protocol)

1. 서 론

CRC (Cyclic Redundancy Codes)는 데이터 전송과정에서 발생하는 오류를 검출하기 위한 기술이다[1][2]. CRC와 유사한 개념으로 이진 데이터에서 1의 개수를 홀수 또는 짝수로 미리 약속하고 이를 만족할 수 있게 한 비트(bit)의 0 또는 1을 추가해서 데이터를 전송하는 패리티 비트(parity bit) 방법이 있으며[2], 데이터를 일정 단위로 자르고 각각을 숫자로 간주한 다음 이를 더한 합을 데이터에 추가하여 오류 유무를 판단하는 체크섬(checksum)도 있다. 또한 오류 검출뿐만 아니라, 이를 제거까지 할 수 있는 오류정정부호(error correcting code)도 있는데, 널리 사용되는 (255,223) RS (Reed Solomon) 부호의 경우에는 223 바이트(bytes)의 데이터에 32

접수일(2014년 9월 5일), 수정일(1차 : 10월 17일), 게재 확정일(2014년 11월 1일)
*탑재체전자팀/{sbseo,yskim1203,pje,kjp123,ssyong}@kari.re.kr, **위성탑재체실, shlee@kari.re.kr

바이트의 RS 오류정정부호를 추가함으로써 255 (223+32) 바이트 데이터 전송 중 발생하는 16개의 오류를 검출하여 제거할 수 있다[3].

패리티 비트 및 체크섬은 계산이 단순하고 부가되는 정보가 데이터에 비해 짧은 장점이 있으나 단순한 덧셈에 기반하므로 오류검출 성능이 우수하지 않다. 반면 오류정정부호는 계산이 복잡하고 부가 데이터가 길지만 오류를 검출뿐만 아니라 제거까지 가능한 특징이 있다. 본 논문의 주제인 CRC의 경우 체크섬과 오류정정부호 사이의 복잡도와 성능을 가진다.

이상의 설명에서, 패리티, 체크섬, CRC, 오류정정부호는 각각 계산하는 방법과 용도가 다르지만, 종종 명칭을 혼용해서 사용한다. 예를 들어 CRC 계산결과를 체크섬이라고 칭하기도 한다.

정지궤도환경탑재체(GEMS, Geostationary Environment Monitoring Satellite)는 2014년 10월 현재 한국항공우주연구원과 미국의 볼 에어로스페이스사(Ball Aerospace & Technologies Corp.)가 공동개발 중에 있으며, 2018년 말에 발사가 계획되어 있는 정지궤도복합위성(GEO-KOMPSAT 2B)에 탑재되어 하루 8회 이상 한반도 주변의 환경을 감시할 예정이다. 정지궤도환경탑재체에서 관측한 데이터를 위성으로 전송할 때 SpaceWire 통신의 GRDDP (GOES-R Reliable Data Delivery Protocol)를 따르며, 해당 GRDDP에서 여덟 비트의 CRC인 CRC8이 사용된다[4].

CRC의 원리는, 미리 정의된 제수(divisor)로 데이터(dividend)를 나눈 후 그 나머지(remainder)를 데이터와 함께 전송하는 것이다[1]. 즉 나머지를 데이터와 함께 전송함으로써, 얻는 장점은 오류 검출이 가능하다는 점이고, 단점은 계산이 필요하고 전송할 데이터의 양이 증가하는 것이다.

또한 덧셈 및 뺄셈에 비해서 나눗셈이 가지는 특징은 (곱셈과 마찬가지로) 이진 시프트(shift) 연산으로 쉽게 구현할 수 있고, 상위 숫자(MSB, Most Significant Byte/Bit)로부터 하위 숫자(LSB, Least Significant Byte/Bit)로 순차 처리가 가능한 것이다.(순차 처리는 데이터가 입력되는 순서와 같다.) 또한 MOD (modulo) 및 XOR (Exclusive OR) 연산을 계산에 도입하여, 직접 산술 나눗셈을 적용하는 경우보다 간단한 방법으로 CRC 계산을 구현할 수 있다.

본 논문은 "A Painless Guide to CRC Error Detection Algorithm[1]" 문서를 근간으로 하여, 정지궤도환경탑재체 CRC8의 개발 및 소프트웨어를 이용한 검증에 필요한 내용을 기술한다.

본 논문의 구성은, 2장에 배경지식을, 3장에 CRC의 기본원리인 MOD 기반 이진 나눗셈 및 레지스터(register)를 이용하는 원리에 대해서 설명한다. 다음으로 4장에서는 CRC를 소프트웨어로 계산하는 방법 및 정지궤도환경탑재체에서 사용하는 CRC8에 대한 설명을, 5장에서는 CRC8의 계산 결과 및 검증을 기술한다.

2. 배경지식

본 논문의 이진 나눗셈을 이해하기 위해서는 MOD 기반 연산, 시프트, 논리연산 XOR 등에 대한 배경지식이 필요하다.

2.1 MOD 기반 연산

십진수(예, 3), 이진수($3 = 11_2$), 팔진수($3 = 3_8$), 십육진수(3 = 0x03)의 보편적인 계산에서는 각 자리의 계산 결과가 윗자리 및 아랫자리의 값에 영향(overflow 또는 underflow)을 줄 수 있다. 반면, MOD 연산에서는 각 자리가 독립적이므로 자리 올림이나 내림이 무시된다. 이진 데이터에 대해서 사칙연산의 덧셈과 MOD 기반의 덧셈의 차이는 다음의 예와 같다.

사칙연산 덧셈: $0011_2 + 0001_2 = 0100_2$
MOD 덧셈: $0011_2 + 0001_2 = 0010_2$

2.2 논리연산 XOR

MOD 기반 연산에서는 자리 올림 및 내림이 무시되기 때문에 덧셈과 뺄셈의 결과가 동일하

며, 아래의 예와 같이 논리연산 XOR (⊕, 이하 XOR)의 결과와도 일치한다.

MOD 기반 덧셈: $0110_2 + 0001_2 = 0111_2$
MOD 기반 뺄셈: $0110_2 - 0001_2 = 0111_2$
논리연산 XOR: $0110_2 \oplus 0001_2 = 0111_2$

따라서 MOD 기반 연산에서 덧셈과 뺄셈은 XOR을 이용해서 계산한다. 여기에서 한 가지 흥미로운 사실은 0110_2를 1_2 증가 시킨 결과와 1_2 감소시킨 값이 일치하므로, 0110_2과 0111_2 중에서 어떤 값이 더 큰 것인지 판단할 수 없다. 하지만 해당 연산에는 자리 올림이 없으므로, 최상위에 존재하는 1_2의 위치를 이용하여 값의 대소를 다음의 예와 같이 판단할 수 있다.

$0001_2 < 0010_2 < 0110_2$

또한 어떤 값에 0_2을 XOR해도($0110_2+0_2 = 0110_2-0_2 = 0110_2 \oplus 0_2 = 0110_2$) 해당 값이 변하지 않음도 참고하면, 이후 알고리즘 이해에 도움이 될 것이다.

2.3 시프트 및 XOR을 이용한 데이터 구성

그림 1과 같이 0111010110_2은 11_2의 시프트와 XOR로 구성할 수 있다[1].

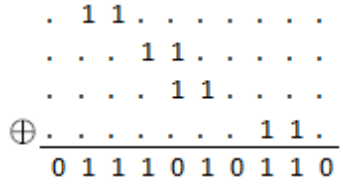

그림 1. 특정 값의 시프트와 XOR을 이용한 데이터 구성의 예

즉, 특정한 값(11_2)에 대해서 시프트와 XOR을 이용해서 생성할 수 있는 값(0111010110_2)이 있고 아닌 경우(예, 0111010111_2)도 있는데, 값을 생성할 수 있는 경우가 (3장에서 설명할) CRC 나눗셈의 기본 원리이다.

2.4 제수의 다항식 표현

CRC 나눗셈에서 제수를 다항식(polynomial)을 이용해서 표현한다[1]. 제수 10011_2은 다항식 $x^4 + x + 1$로, 다항식의 길이 W는 (5가 아닌) 4로 정의한다. 또 다른 표현 방법으로는, 제수의 최상위비트는 항상 1이므로 이를 제외하고 나머지를 십육진수인 0x03으로 표현한다.

3. CRC 나눗셈

본 장에서는 CRC 나눗셈과(3.1절), 이를 레지스터를 이용해서 구현하는 방법(3.2절)을 상세히 설명한다.

3.1 CRC 나눗셈

이진수에 대한 CRC의 나눗셈은 데이터의 뒷부분에 다항식의 길이 W 만큼의 0_2을 추가하고, 이를 MOD 기반 나눗셈으로 계산하는 것으로, 그림 2에 예를 들어 설명하였다.

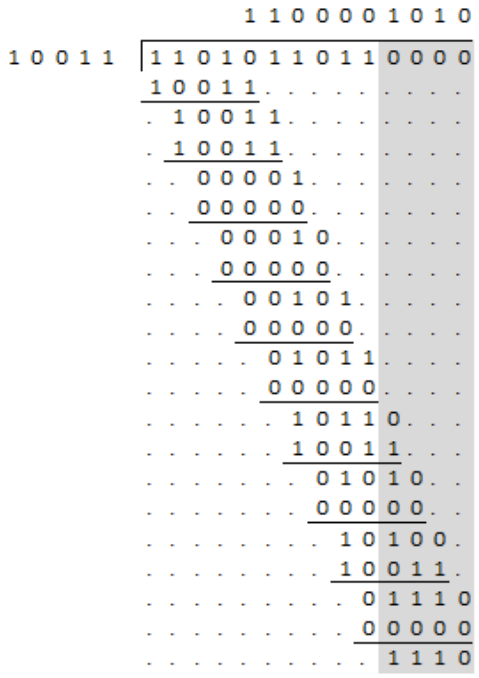

그림 2. 이진수의 CRC 나눗셈

그림 2에서 4 개의 0_2을 데이터 뒤쪽에 추가하였으며(회색 네 비트), 이를 다항식 10011_2로 나누었을 때 나머지는 1110_2이다.

여기에서 나머지는 네 자리 숫자(W=4)로 다항식(다섯 자리)보다 작으며, 11010110110000_2은 다항식으로 나눠지지 않지만 해당 값에 나머지를 더한 11010110111110_2은 나눠떨어진다. 즉, 11010110111110_2은 그림 3과 같이 다항식의 시프트와 XOR로 조합이 가능한 값이다.

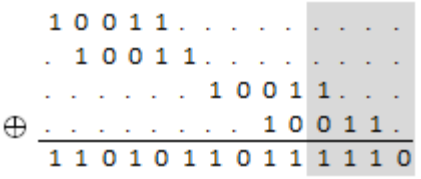

그림 3. 다항식의 시프트와 XOR를 이용한 데이터 생성

3.2 레지스터를 이용한 CRC 나눗셈

레지스터를 이용해서 CRC 나눗셈을 구현하는 방법을 그림 4에 설명하였다.

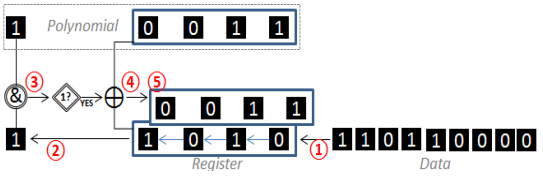

그림 4. 레지스터를 이용한 CRC 나눗셈(비트)

그림 4에서 레지스터는 먼저 들어간 데이터가 먼저 나오는(FIFO, First In First Out) 큐(que)이며, 데이터가 입력①되어 레지스터에서 밀려난② 데이터의 값에 따라서③ 다항식과 XOR 연산④을 수행하여 레지스터 값을 갱신⑤한다. 즉, 레지스터에서 나온 값은 데이터 입력 직전 레지스터의 최상위 비트이며 (데이터 입력 직전 레지스터의 나머지 세 비트는 한 비트씩 좌측으로 이동하여 레지스터에 저장되어 있음), 다항식에도 최상위 비트인 1_2이 생략되었기 때문에 이 둘이 모두 1_2일 때 XOR 연산이 수행되는 것이다.(결국, 레지스터에서 밀려난 값이 XOR 연산 여부를 결정한다.) 즉, 그림 2 및 그림 3의 연산과정을 그림 4에서는 레지스터를 이용해서 구현한 것이라고 이해할 수 있다.

표 1은 그림 4의 레지스터 값의 변화를 시프트에 따라서 설명한 내용이다.

표 1. 레지스터를 이용한 CRC 나눗셈에서의 값 변화 (다항식: 0x03)

Shift	Reg^2	Pop	Reg^1	Data
0	NA	NA	0000_2	11010110110000_2
1	0001_2	0_2	0001_2	$.1010110110000_2$
2	0011_2	0_2	0011_2	$..010110110000_2$
3	0110_2	0_2	0110_2	$...10110110000_2$
4	1101_2	0_2	1101_2	$....0110110000_2$
5	1001_2	1_2	$101\ 0_2$	$.....110110000_2$
6	0000_2	1_2	0011_2	$......10110000_2$
7	0001_2	0_2	0001_2	$.......0110000_2$
8	0010_2	0_2	0010_2	$........110000_2$
9	0101_2	0_2	0101_2	$.........10000_2$
10	1011_2	0_2	1011_2	$..........0000_2$
11	0101_2	1_2	0110_2	$...........000_2$
12	1010_2	0_2	1010_2	$............00_2$
13	0111_2	1_2	0100_2	$.............0_2$
14	1110_2	0_2	1110_2	$..............$

표 1과 그림 2를 비교해서 살펴보면 그 계산과정이 서로 일치함을 알 수 있다. 표 1 및 그림 2에서 다항식은 0x03이며, 표 1의 Shift 4 계산 결과(Reg^2)로 부터 Shift 5까지의 계산과정을 설명하면 다음과 같다.

- 레지스터로 0_2이 입력되면(Shift 5, Reg^1; 0_2)
- 레지스터 값이 1101_2(Shift 4, Reg^2)에서 1010_2(Shift 5, Reg^1)으로 변경되고
- 1_2이 레지스터에서 나온다.(Shift 5, Pop)
- 레지스터에서 나온 값이 1_2이므로, 현재 레지스터 값 1010_2은 다항식 0011_2과 XOR 연산을 수행하여,
- 결과인 1001_2가 레지스터에 저장된다.(Shift 5, Reg^2)
- (참고) Shift 5에서 레지스터 값이 1001_2이므로(최상위 비트가 1_2이다), Shift 6에서도 레지스

터와 다항식의 XOR 연산이 수행됨을 예상할 수 있다.

4. 테이블 기반 CRC

본 장에서는 CRC 소프트웨어 구현을 위해서 CRC 나눗셈을 비트에서 바이트 기반으로 확장하고(4.1절), 데이터를 레지스터에 통과시켜 처리하는 기존 방식을 개선한 테이블 기반 CRC인 Direct Table Algorithm에 대해서 기술한다.(4.2절) 다음으로 실제 CRC를 활용할 때 설정해야 하는 '다항식, 레지스터 초기값, Final XOR'에 대해서 논의한 후(4.3절), 마지막으로 현재 정지궤도환경탑재체에서 정지궤도복합위성으로 데이터를 전송하는 GRDDP에서 사용하는 CRC8 계산법을 코드를 이용해서 설명한다. (4.4절)

4.1 테이블 기반 CRC 나눗셈

그림 4에서 한 비트의 데이터가 레지스터로 들어오면, 레지스터의 최상위 비트가 밀려나고 밀려난 값이 1_2일 때 레지스터와 다항식의 XOR이 수행되었다.

그림 4의 레지스터는 네 비트인데, 바이트 단위의 처리를 설명하기 위해서 먼저 레지스터를 네 바이트로 증가시킨다. 증가된 네 바이트의 레지스터에 대해서 동일하게 한 비트 씩 8회 데이터를 입력 시키면, 한 바이트가 레지스터로 입력되는 것과 동일한 동작이 된다.

네 바이트의 레지스터에서 8회에 걸쳐 한 비트씩 입력하고 난 다음에, 밀려나는 값의 종류는 00000000_2부터 11111111_2까지 총 256이다. 여기에서 한 비트씩 밀려나는 과정(시프트)에서 해당 비트가 1인 경우에는 (CRC 나눗셈을 위해서) 각각 비트에 대해서 XOR을 수행해야하는데, XOR에서 교환법칙([[A⊕B]⊕C]⊕D=A⊕[B⊕C⊕D])이 성립하므로 다항식에 대해서 8회의 시프트 및 XOR을 수행한 다음([B⊕C⊕D]) 마지막에 레지스터 값(A)과 XOR을 취해도 레지스터에 순차적으로 XOR한 결과와 동일하다.(시프트 및 XOR을 이용해서 데이터를 생성하는 그림 3과 동일한 상황)

따라서 다항식에 대한 256 종류의 시프트 및 XOR 값을 미리 계산해 두고, (한 비트가 아닌) 한 바이트가 밀려났을 때 그 값을 LUT (Look Up Table, 이하 테이블)로 참조해서 사용하면 바이트 단위 처리가 가능하다. 그림 5에 미리 계산한 테이블을 이용하여 한 바이트 단위로 CRC 나눗셈을 수행하는 과정을 설명하였다.

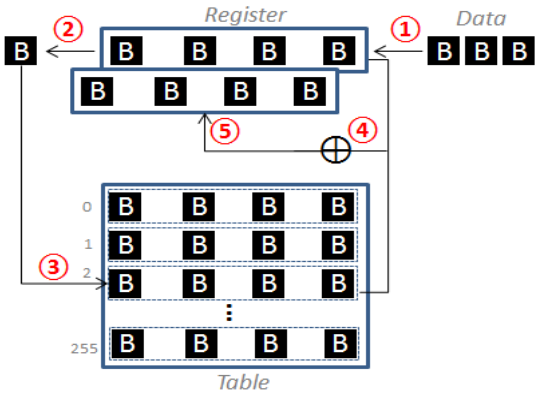

그림 5. 레지스터를 이용한 CRC 나눗셈(바이트)

그림 5에서 한 바이트가 레지스터로 입력되면 ① 한 바이트가 밀려나며② 미리 계산한 테이블에서 이 값에 해당하는 위치(그림 5에서는 '2')의 값③을 현재 레지스터의 값과 XOR한 다음 ④ 그 값으로 레지스터 값을 갱신한다⑤.

그림 5에서 사용하는 테이블은 다항식, 시프트, 그리고 XOR을 이용해서 계산한다. 한 바이트에서 나올 수 있는 256 종류의 테이블을 t[i]라고 할 때(i = 0~255), 다음의 코드로 레지스터를 이용한 CRC 나눗셈에 사용되는 테이블을 계산할 수 있다.

```
FOR i = 0 to 255
  t[i] = i
  FOR j = 0 to 7
      IF (t[i]'s MSB == 1)
```

```
        t[i] = (t[i]<<1) XOR (POLY)
      ELSE
        t[i] << 1
      END_IF
    END_FOR
  END_FOR
```

코드에서 t[i]이 계산하고자 하는 결과인 테이블 값인데, 0부터 255까지 각각의 값에 대해서 8회(FOR j = 0 to 7) 씩의 루프를 돌며, 최상위비트(MSB)가 1일 때는 한 비트 시프트한 다음 다항식(POLY)과 XOR을 수행하고(IF (t[i]'s MSB == 1)), 최상위비트가 1이 아닌 경우(ELSE)에는 시프트만 수행한다.

레지스터를 이용한 CRC 나눗셈에서, 그림 4(비트)의 경우 레지스터 값이 직접 다항식에 XOR 되는 반면, 그림 5(바이트)에서는 다항식으로 미리 테이블 값을 계산해 두고 레지스터 밖으로 밀려나온 값으로 테이블 참조한 다음 그 값을 레지스터와 XOR하는 차이가 있다.

즉, 그림 5에서 CRC 나눗셈을 수행하려면 전체 데이터가 레지스터를 통과해서 밖으로 밀려나야만 그 값으로 테이블 값을 결정하고 해당 값과 레지스터 값을 XOR 할 수 있다. 따라서 그림 5의 레지스터가 네 바이트이므로(W=32) CRC 나눗셈을 수행하기 위해서는 데이터 뒷부분에 총 여덟 바이트(W÷8×2)의 0x00이 추가되어야 한다. 전반 네 개의 0x00은 데이터를 레지스터까지 밀어 넣는 역할을 하고 '후반 네 개의 0x00'은 '전반 네 개의 0x00'(그림 2에서 데이터 뒷부분에 붙은 네 개의 0_2과 동일한 의미로, 나머지가 저장되는 공간에 해당한다)을 레지스터 밖으로 밀어내는 역할을 한다.

또한 데이터가 입력되기 전에 레지스터에는 초기값이 채워져 있는데, 데이터가 레지스터 밖으로 밀려나는 동작이 수행되기 전에 레지스터 값이 모두 밖으로 밀려난다.(앞서 설명한 8개의 0x00 중 전반 네 개에 의해 수행됨) 이 때 레지스터 값이 모두 0x00이면 레지스터에 데이터가 채워지는 동작만 하고 (0x00이 연속적으로 밀려나오므로 XOR은 수행되지 않는다. CRC 나눗셈과 동일), 만약 레지스터 초기값이 0x00이 아닌 경우에는 레지스터와 다항식의 XOR이 수행되어 그 값으로 레지스터가 갱신됨을 알 수 있다.(CRC 나눗셈과 다름. 4.3절에서 설명)

4.2 Direct Table Algorithm

테이블 기반 CRC에서 나눗셈을 이용할 경우 레지스터 초기값을 밀어내는 과정에서 시프트 연산과 XOR, 그리고 데이터를 레지스터 밖으로 완전히 밀어내기 위한 과정을 고려해야 함을 확인하였다.

반면 Direct Table Algorithm (테이블 기반 알고리즘)은 앞서 설명한 테이블 기반 CRC 나눗셈을 개선한 방법으로[1], 데이터가 레지스터를 통과하지 않고 동작한다. 따라서 초기값이나 데이터를 레지스터에서 밀어내는 과정을 별도로 수행하지 않아도 되며(계산에 이미 포함되어 있음), 데이터가 (레지스터 통과 없이) 즉시 계산에 이용된다. 그림 6에 Direct Table Algorithm의 동작을 여섯 가지로 구분하여 설명한다.

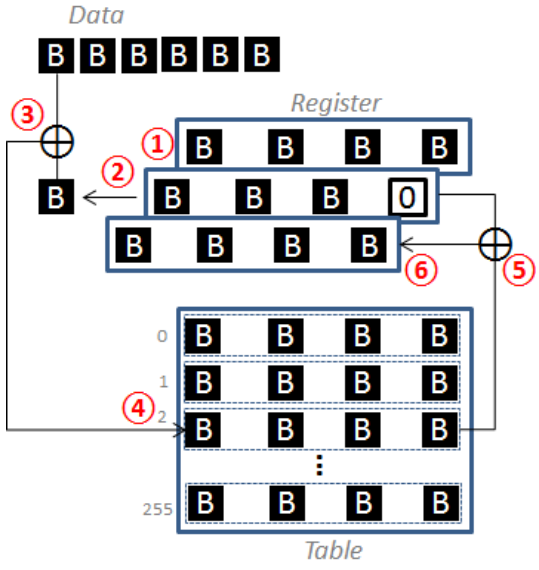

그림 6. Direct Table Algorithm

먼저 레지스터를 한 바이트 시프트하면서 최하위 바이트는 0x00으로 채운다①. 다음으로 레

지스터에서 밀려나온 바이트②를 데이터와 XOR하고③ 이에 대응되는 Table 값④을 레지스터와 XOR한 다음⑤ 그 값을 레지스터에 갱신한다⑥.

그림 6의 계산에서, 초반 네 번의 시프트①에 의해서 레지스터 내의 초기값은 모두 레지스터 밖으로 밀려나며, 데이터의 경우 레지스터를 통과하지 않고 직접 모두 사용한다. 따라서 테이블 기반 CRC 나눗셈보다 간편하게 CRC 계산을 수행할 수 있다. Direct Table Algorithm에서 사용하는 테이블은 레지스터를 이용한 CRC 나눗셈(바이트)의 t[i]와 동일하다.(4.1절 참고)

Direct Table Algorithm 동작을 코드로 기술하면 다음과 같다.(레지스터의 길이는 방정식으로 결정된다. 예를 들어 네 바이트 길이의 레지스터의 경우 W=32임을 알 수 있다.)

```
WHILE (d = read 1 byte from DATA)
        r = (r<<8) XOR t[(r>>(W-8)) XOR d]
END_WHILE
```

위의 코드의 동작은, 데이터(DATA)를 한 바이트씩 읽은 다음(d)③, 이 값(d)과 레지스터의 최상위 바이트(r>>(W-8))②를 XOR 계산한 결과③로 테이블(t[]) 값을 결정하고(t[(r>>(W-8)) XOR d])④ 이를 최하위 바이트에 0x00을 채운 레지스터 값(r<<8)①과 XOR을 수행한 후⑤ 이 값으로 레지스터를 갱신(r)하는 것이다⑥.

4.3 다항식, 레지스터 초기값, Final XOR

지금까지 MOD 연산, XOR의 원리에서 시작하여 CRC를 구현할 수 있는 Direct Table Algorithm까지 설명하였다. 서론의 첫 부분으로 잠시 돌아가서, CRC의 정의를 다시 살펴보면 아래와 같다.

CRC는 데이터 전송과정에서의 오류를 검출하는 목적을 위해서 원 데이터에 추가하는 정보이다.

CRC 계산을 위해서 CRC 나눗셈을 이용할 수도 있고 마지막으로 살펴본 Direct Table Algorithm을 적용할 수도 있으며, 또는 본 논문에서 설명하지 않은 다른 방법도 가능할 것이다. 만약 CRC 나눗셈을 이용할 경우, 나눗셈의 과정을 엄밀히 지켜서 CRC를 계산하고자 한다면 레지스터 초기값은 반드시 모두 0x00으로 설정해야 한다.(4.1절 참고) 만약 레지스터 초기값을 0x00이 아닌 다른 값으로 설정해서 계산한 CRC는 어떤 의미가 있을까? 이 경우 CRC가 나눗셈에서의 나머지와 일치하지는 않지만, CRC 결과를 '데이터 전송과정 중 오류를 검출하는 CRC의 목적'에 활용할 수 있다는 것에는 변함이 없다. 단, 전제는 데이터를 수신하는 쪽에서도 레지스터의 초기값이 무엇이었는지는 알고 있어야한다.

또한 계산된 최종 CRC에 미리 정의된 어떤 값(Final XOR)을 XOR 해서 CRC 계산을 마무리 할 수 도 있다. 이 경우 역시 Final XOR을 수신측에서 알고 있다면 이 역시 CRC 목적에 합치된다. 본 논문의 목적인 (정확한 나눗셈이 아니라) CRC 계산이라는 관점에서 생각하면, 레지스터 초기값과 Final XOR은 어떤 값이든지 미리 결정만 한다면 문제가 되지 않는다.

미리 결정(정의)해야 할 값 (즉, CRC 계산하는 곳과 수신측에서 동시에 알아야할 값) 중에 레지스터 초기값과 Final XOR 보다 더 중요한 것이 방정식이다. 이 방정식을 어떻게 결정하는가에 따라서 오류 검출 성능이 달라지는데, 이 값은 데이터 및 전송 채널(channel)의 특성을 고려하여 결정한다.

CRC 계산 관점에서 본 절을 정리하면, '방정식, 레지스터 초기값, Final XOR' 세 가지를 미리 정의하면 CRC를 그 목적에 맞게 활용할 수 있다. 예를 들어, 비트 단위에서 특정 값이 연속적으로 존재하는 데이터의 경우 레지스터 초기값을 0x00이 아닌 다른 값을 사용하는 것이 데이터 경계를 확인하는데 유리할 수 있음을 예상할 수 있다.

4.4 정지궤도환경탑재체의 CRC8

정지궤도환경탑재체와 정지궤도복합위성 간의 GRDDP에는 CRC8이 사용된다[2]. CRC8에 대한 방정식, 레지스터 초기값, Final XOR은 다음과 같으며, 이후 설명하는 코드로 계산할 수 있다.

- CRC8 방정식 : 0x07 ($x^8 + x^2 + x + 1$)
- CRC8 레지스터 초기값 : 0xFF
- CRC8 Final XOR : 0x00

```
FOR i = 0 to 255
  t[i] = i
  FOR j = 0 to 7
      IF (t[i] AND 0x80)
        t[i] = (t[i]<<1) XOR 0x07
      ELSE
        t[i] << 1
      END_IF
  END_FOR
END_FOR
```

CRC8에서 테이블(t[i])과 입력값(d)을 이용한 CRC8 계산(r)은 다음과 같이 수행할 수 있다.

```
WHILE (d = read 1 byte from DATA)
      r = t[r XOR d]
END_WHILE
```

계산이 간단해진 이유는, 레지스터(r)가 한 바이트(W=8)이기 때문에 'r<<8'은 항상 0x00이며, 따라서 4.2절의 Direct Table Algorithm 식에서 0x00과의 XOR인 '(r<<8) XOR'은 무시할 수 있다. 그리고 최상위 바이트를 취하는 'r>>(W-8)' 계산 역시 필요하지 않고 r값을 직접 사용하면 된다.(r>>0 = r)

위의 코드의 동작은, 데이터를 한 바이트씩 읽어온 다음(d) 레지스터(r)와 XOR한 값을 테이블에 찾아서(t[r XOR d]) 그 값으로 레지스터를 갱신시키는 것(r = t[r XOR d])을 데이터 바이트 수만큼 반복해서 수행하는 것이다. 따라서 다항식(0x07)에 따른 테이블 값(t[i])만 준비한다면 CRC8은 간단하게 계산(r)할 수 있음을 알 수 있다.

5. CRC8 계산 결과 및 검증

본 장에서는 정지궤도환경탑재체에서 사용하는 CRC8 계산 결과 및 이를 검증한 내용에 대해서 설명한다[5].

5.1 CRC8 테이블 생성 결과

정지궤도환경탑재체 GRDDP의 CRC8에 대한 256개의 테이블 값(t[i]) 계산결과는 그림 7과 같다. 그림 7에서 괄호 내의 숫자는 해당 행의 첫 번째 열의 순서를 기입한 것으로, 만약 t[212]에 해당하는 값은 (210) 행에서 세 번째인 0x22이다.

```
(000) 0x00,0x07,0x0E,0x09,0x1C,0x1B,0x12,0x15,0x38,0x3F,
(010) 0x36,0x31,0x24,0x23,0x2A,0x2D,0x70,0x77,0x7E,0x79,
(020) 0x6C,0x6B,0x62,0x65,0x48,0x4F,0x46,0x41,0x54,0x53,
(030) 0x5A,0x5D,0xE0,0xE7,0xEE,0xE9,0xFC,0xFB,0xF2,0xF5,
(040) 0xD8,0xDF,0xD6,0xD1,0xC4,0xC3,0xCA,0xCD,0x90,0x97,
(050) 0x9E,0x99,0x8C,0x8B,0x82,0x85,0xA8,0xAF,0xA6,0xA1,
(060) 0xB4,0xB3,0xBA,0xBD,0xC7,0xC0,0xC9,0xCE,0xDB,0xDC,
(070) 0xD5,0xD2,0xFF,0xF8,0xF1,0xF6,0xE3,0xE4,0xED,0xEA,
(080) 0xB7,0xB0,0xB9,0xBE,0xAB,0xAC,0xA5,0xA2,0x8F,0x88,
(090) 0x81,0x86,0x93,0x94,0x9D,0x9A,0x27,0x20,0x29,0x2E,
(100) 0x3B,0x3C,0x35,0x32,0x1F,0x18,0x11,0x16,0x03,0x04,
(110) 0x0D,0x0A,0x57,0x50,0x59,0x5E,0x4B,0x4C,0x45,0x42,
(120) 0x6F,0x68,0x61,0x66,0x73,0x74,0x7D,0x7A,0x89,0x8E,
(130) 0x87,0x80,0x95,0x92,0x9B,0x9C,0xB1,0xB6,0xBF,0xB8,
(140) 0xAD,0xAA,0xA3,0xA4,0xF9,0xFE,0xF7,0xF0,0xE5,0xE2,
(150) 0xEB,0xEC,0xC1,0xC6,0xCF,0xC8,0xDD,0xDA,0xD3,0xD4,
(160) 0x69,0x6E,0x67,0x60,0x75,0x72,0x7B,0x7C,0x51,0x56,
(170) 0x5F,0x58,0x4D,0x4A,0x43,0x44,0x19,0x1E,0x17,0x10,
(180) 0x05,0x02,0x0B,0x0C,0x21,0x26,0x2F,0x28,0x3D,0x3A,
(190) 0x33,0x34,0x4E,0x49,0x40,0x47,0x52,0x55,0x5C,0x5B,
(200) 0x76,0x71,0x78,0x7F,0x6A,0x6D,0x64,0x63,0x3E,0x39,
(210) 0x30,0x37,0x22,0x25,0x2C,0x2B,0x06,0x01,0x08,0x0F,
(220) 0x1A,0x1D,0x14,0x13,0xAE,0xA9,0xA0,0xA7,0xB2,0xB5,
(230) 0xBC,0xBB,0x96,0x91,0x98,0x9F,0x8A,0x8D,0x84,0x83,
(240) 0xDE,0xD9,0xD0,0xD7,0xC2,0xC5,0xCC,0xCB,0xE6,0xE1,
(250) 0xE8,0xEF,0xFA,0xFD,0xF4,0xF3
```

그림 7. 정지궤도환경탑재체 CRC8 계산 결과

5.2 CRC8 계산 결과

두 가지 입력([i], [ii])에 대한 CRC8의 계산결과는 다음과 같다.

[i] 0x00 00 00 00 : 0xD1
[ii] 0x29 EE 33 01 00 00 29 01 : 0xDE

두 번째 입력([ii])은 여덟 바이트로 구성되었으므로, CRC 계산에서 레지스터 값은 0xFF를 시작으로 0xDE까지 총 8회 갱신되며 변화는 다음과 같다.

0xFF → 0x2C → 0x40 → 0x5E → 0x9A → 0xCF → 0x63 → 0xF1 → 0xDE

5.3 CRC8 계산 결과 검증

5.2절의 두 개의 계산결과는, CRC8을 생성하는 정지궤도환경탑재체 개발자와 CRC8을 사용하는 정지궤도복합위성 개발자의 확인을 완료하였다.(GEMS-SC-IF-0014A, 2014.07.31, 보안문서로 공개불가) 그리고 그림 8의 GUI (Graphic User Interface)와 같이, 웹페이지에서도 CRC8 계산값의 유효성을 다음과 같이 확인할 수 있다.

<웹페이지 주소>
http://www-user.tu-chemnitz.de/~heha/viewzip.cgi/mb-iwp/DDS130/DDS130.zip/crc.html

<입력하는 값>
CRC order (1.. 64) : 8
CRC polynom (hex): 07
Initial value (hex): FF, direct
Final XOR value (hex): 00
Bit reversing: No check
Data sequence: %29%EE%33%01%00%00%29%01

<결과>
Result: DE (hex), 8 bytes

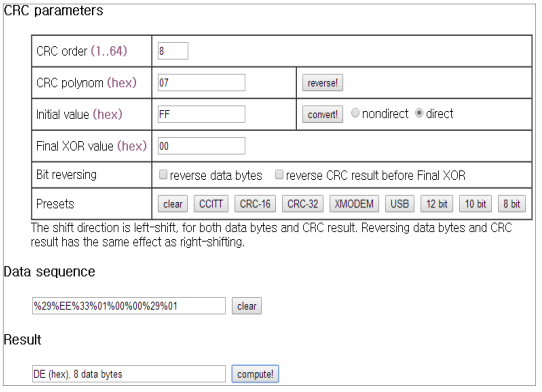

그림 8. CRC 계산결과 확인 소프트웨어 GUI

6. 결 론

본 논문에서는 CRC의 배경이론부터 시작하여, CRC의 원리 및 업무에 직접 활용할 수 있는 테이블 기반 CRC의 방법인 Direct Table Algorithm의 설명, 구현 및 검증을 기술하였다.

정지궤도환경탑재체에서 위성으로 관측영상을 전송할 때 (GRDDP에서 정의된) CRC8을 사용하며, 이때 방정식은 0x07 ($x^8 + x^2 + x + 1$), 레지스터 초기값은 0xFF, Final XOR은 0x00이다.

본 논문에서는, 정지궤도환경탑재체의 CRC8을 Direct Table Algorithm으로 처리하는데 직접 활용할 수 있는 256개의 테이블 값을 계산하여 정리하였고(5.1절), 계산방법을 코드 형태로 기술하였으므로(4.4절), 이를 이용하여 CRC8을 계산하는 소프트웨어를 간단하게 제작할 수 있을 것으로 기대한다. 계산결과는 5.3절에 소개한 웹페이지에서 검증이 가능하다.

참 고 문 헌

1. Ross N. Willians, "A Painless Guide to CRC Error Detection Algorithm", http://www.zlib.net/crc_v3.txt, (2014년 10월 15일 접속 확인)
2. Peterson, William Wesley, and Daniel T. Brown. "Cyclic codes for error detection." Proceedings of the IRE 49.1 (1961): 228-235.
3. Wikipedia, “Reed-Solomon error correction”,http://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction, (2014년 10월 15일 접속 확인)
4. NASA, “GOES-R Reliable Data Delivery Protocol (GRDDP)”, NASA Goddard Space Flight Center, Document No: 417-R_RPT-0050, RM Version, 2008.
5. 서석배, 김영선, 박종억, 공종필, 용상순, “Generation Method of Table Driven CRC8 for the GOES-R Reliable Data Delivery Protocol”, 한국우주과학회 가을학술대회, 2014.